Universidad Autónoma de Madrid

Escuela Politécnica Superior

Doble Grado En Ingeniería Informática y Matemáticas

TRABAJO DE FIN DE GRADO

USE OF DEEP LEARNING TECHNIQUES ON MEDICAL IMAGES

Alfonso Hernandez Silva
Tutor: David Camacho

Enero 2019

USE OF DEEP LEARNING TECHNIQUES ON MEDICAL IMAGES

Autor: Alfonso Hernandez Silva
Tutor: David Camacho

Departamento de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid

Enero 2019

Agradecimientos

Quiero agradecer a mis padres por haberme apoyado para la realización de este trabajo y todos estos años en la Universidad que han sido necesario para ello. También quiero agraderselo a mi compañera Helena, así como a mis amigos y al resto de mi familia. Por último quiero agradecérselo a David y al resto de miembros del grupo de Investigaión AIDA por ayudarme con el proyecto.

I want to thank my parents for having supported me for the realization of this work and all these years in the University that have been necessary for it. I also want to thank my partner Helena, as well as my friends and the rest of my family. Finally I want to thank David and the other members of the AIDA Research group for helping me with the project.

Abstract

In recent years the world of artificial intelligence has undergone a great expansion, thanks to the advances in Machine Learning and Deep Learning. One of the advances that has been achieved is the design of algorithms capable of automatically classifying images. This work will focus on Supervised Learning's Algorithms for the recognition of Images.

One of the main characteristics of these algorithms is that they need a very large amount of information to learn how to classify correctly the desired images, i.e, they need a large number of labeled images. However, in many cases the set of labeled images is not large enough to train a model optimally, as in the case of medical images such as scanners or radiography, since access to data is usually very complicated.

That is why this work has focused on the study of different algorithms for the classification of radiographs of healthy patients and patients with Tuberculosis. When diagnosing Tuberculosis, if the patient is suspect of suffering from Tuberculosis, as with all infectious diseases the only way to confirm the diagnosis is to realize a culture, which is specially long with tuberculosis (more than one month), so our goal is to look for a model capable of reduce the total amount of cultures needed, by confirming with a high statistical evidence the existence or not of tuberculosis in the patient.

To achieve this, first a theoretical study has been carried out, with a deep analysis both of the different techniques to represent the image in the computer and the necessary processing of the images so that they are ready to be managed by the algorithms, as well as of the different algorithms and their components.

Then, several experiments have been carried out to analyze the performance of different Deep Learning algorithms, such as several types of Convolutional Neural Networks and other Machine Learning algorithms, such as Support Vector Machines. Then, we have tried to combine the different algorithms to create a new one that improves the model effectiveness, with the main goal to reduce the number of False Positives and/or Negatives to 0. Finally, the value of a model with that results to help in tuberculosis diagnose has been analyzed.

Key words — Machine Learning, Deep Learning, Image Classification, Convolutional Neural Networks, Support Vector Machines

Resumen

En los últimos años el mundo de la Inteligencia Artificial ha vivido una gran expansión, gracias en parte al desarrollo del Aprendizaje Automático y del Aprendizaje Profundo. Uno de los avances logrados es conseguir desarrollar algoritmos capaces de interpretar una imagen de manera automática. Este trabajo se centrará en los algoritmos de Aprendizaje Supervisado para el reconocimiento de Imágenes.

Una de las principales características de estos algoritmos es que requieren de una cantidad de información muy grande para aprender a clasificar bien las imágenes deseadas, es decir, necesitan una gran cantidad de imágenes ya clasificadas. Sin embargo, en numerosas ocasiones el conjunto de imágenes ya clasificadas no es lo suficientemente grande para entrenar un modelo de manera óptima, como es el caso de imágenes médicas como escáneres o radiografías, ya que el acceso a los datos suele ser muy complicado.

Es por eso que este trabajo se ha centrado en el estudio de diversos algoritmos para la clasificación de radiografías de pacientes sanos y pacientes con Tuberculosis. A la hora de diagnosticar la Tuberculosis, si el paciente es sospechso de padecer tuberculosis, al igual que con todas las enfermedades infecciosas, la única manera de confirmar con seguridad el diagnóstico, es con un cultivo, que en el caso de la tuberculosis es especialmente largo (superior a 1 mes), por lo que nuestro objetivo es buscar un modelo que sea capaz de confirma,r sin error relevante, la existencia o ausencia de la enfermedad en un porcentaje de los pacientes sospechosos de padecerla.

Para ello, primero se ha realizado un estudio teórico, con un analisis tanto de las distintas técnicas para representar y analizar imágenes por computadora y el procesado necesario de las imágenes para que estén listas para introducirse en cualquiera de los algoritmos, como de los distintos algoritmos seleccionados.

Finalmente, se ha diseñado un experimento para analizar el rendimiento de distintos algoritmos de Aprendizaje Profundo (se han seleccionado varios tipos de Redes Neuronales Convolucionales) y algoritmos de Aprendizaje Automático, como las Máquinas de Soporte Vectorial. Por último se ha tratado de combinar los distintos algoritmos para crear uno que aumente la efectividad, buscando reducir a 0 el numero de Falsos Positivos y/o Negativos. Por último se ha analizado el valor práctico que tendría un modelo con esos resultados en el diagnóstico de enfermedades.

Palabras clave — Aprendizaje Automático, Aprendizaje Profundo, Reconocimiento de Imágenes, Redes Neuronales Convolucionales, Máquinas de Soporte Vectorial

Índice general

Índice de tablas

Índice de figuras

1 Introduction

1.1. Project Motivation

This project has two main goals: the first one is to get a better understanding of some of Machine Learning and Deep Learning Algorithms, by the theoretical study of some techniques to process the raw data increasing algorithm's performance. We will also analyze some Machine Learning Algorithms (Support Vector Machines) and Deep Learning Algorithms (Convolutional Neural Networks).

The second goal of the project is to understand how to use these algorithms in a field like medicine, trying to build a model to help in tuberculosis diagnosis, identifying what kind of solution needs the problem, how to measure the performance of every algorithm and how to find the model which fit the best with the dataSet and the requisites.

For this reason, I chose a dataset of pulmonary chest X-Ray images in which half of the images have abnormalities attached to tuberculosis, and the other half are from healthy patients. The project will be routed to the creation of a model capable of helping in tuberculosis diagnosis by avoiding unnecessary tests, which could be dangerous, invasive or simply more expensive, measure its performance and analyze what advantages would the implantation of the model have in a tuberculosis diagnosis service[1].

2

BackGround

2.1. Digital Image Processing

2.1.1. Digital Images

It's necessary to understand that the way the human eye interprets an image is not the same than the way a computer does. We have to find a mathematical way to represent each image if we want to represent them in a computer.

An image may be defined as a two-dimensional function, where x and y are spatial (plane) coordinates, and the amplitude of f at any pair of coordinates (x, y) is called the intensity or gray level of the image at that point. When x, y, and the intensity values of f are all finite (discrete quantities) we call the image a digital image[2]. Then, digital images are composed by a finite number of elements that we call **pixels**, each one with a different location, where they represent the basic unit of digital images.

Therefore, we can represent an image in a computer with a system of coordinates, in which each element is a number that represents the value of the pixel attached to that location.

Pixels

In a digital image, the value of a pixel describes its brightness. The most common range of possible values of a pixel is (0, 255), and that is because each pixel is stored in a byte, i.e, 8-bits, which can take values in that range, where 0 is usually black and 255 is white. This will be the Image Representation we will use for this work, but a pixel could

take a different range of values. For example, for some high definition images pixels have a 12-bit color-depth, i.e, they can take values from 0 to 4095.

Channels

Depending on the type of image it will has one or more channels. A channel on an image is another image of the same size but they only represent a part of the pixel, that depends on the type of image:

- Gray scale Images: They consist in only one channel where each value represents the brightness of the pixel
- RGB Images: They consist in 3 channels, the values of different channels are the brightness of the pixel for each one of the primary color: red 'R', green 'G' and blue 'B'. So each pixel is formed by the combination of this 3 channels, then each pixel is represented by 8*3 = 24 bits
- Multi-espectral Images: They are formed by more than 3 channels, and the possible values of each pixel are usually stored in a colormap, where the color intensity can be looked up [3].

This part is especially important for this project because although we will use gray scale images, some of the algorithms that will be employed were designed for RGB images, i.e, they will expect an image with 3 channels as input. To solve this, we will triple the existing channel to get 3 identical channels.

2.1.2. Processing Data

Once we have stored a mathematical representation of an image in the computer, we can make some operations to our data before passing it to the classification algorithm. There are several techniques for improving the algorithms accuracy.

Histogram Approach

The histogram of a digital image is a distribution of its discrete intensity levels in the range of the possible values of each pixel. The distribution is a discrete function h associating to each intensity level: r_k the number of pixel with this intensity: n_k. Histogram equalization is a method to process images in order to adjust the contrast of an image by modifying the intensity distribution of the histogram. The objective of this technique is to give a linear trend to the cumulative probability function associated to the image. We will use the cumulative probability function and we will modify the image to get a new linear cumulative probability function[4].

Another technique based on Histograms to improve contrast in the image is Contrast Limited Adaptive Histogram Equalization (CLAHE). Instead of computing only one histogram, the CLAHE computes several Histograms, one for each region in the image, improving the local contrast. In addition, it clips the histogram at a predefined value, so it enhances contrast of the image but prevents the noise for being amplified in excess[5].

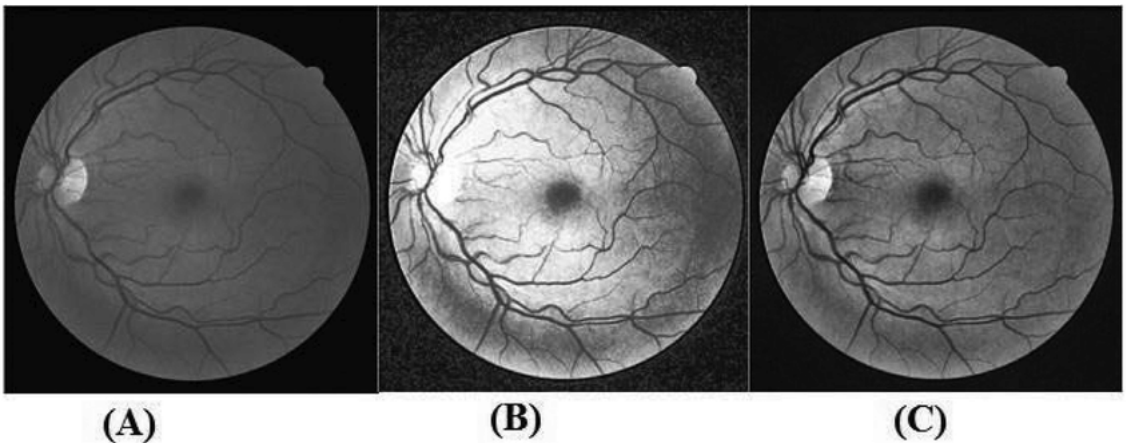

Figura 2.1: Image Processing Based on Histograms. From left to right: Original Image, Image after applying histogram equalization, Image after applying CLAHE

Positional Approach

Another way to improve the model efficiency is to increase the size of the Training Set. One of our objectives when creating a model is that it must be able to correctly classify the images, even when they are in different rotation, angles, positions, under some noise that the images may have, etc. To achieve this we are going to analyze different techniques to increase the size of our training set with images created from the original Training Set:

- Mirroring: For each image in the Training Set we can make other by flipping it, increasing the model robustness especially if the perspective of the image is unknown.
- Translational Shift: This method adds new images to the Training Set that are copies of other images but shifted in some direction.
- Adding Noise: It increases the model robustness against images with some distortion.
- Random Crop: It helps the model ignoring the image background.

For each dataset it is necessary to analyze what kind of data augmentation would be more suitable to improve the overall robustness of the model[6].

Figura 2.2: Image Processing Based on Positional Approach. From left to right: Original Image, Mirror Image, Cropped Image

Statistical Approach

There are several ways to process our data based on statistical analysis and they are usually the most extensive and complex. However, most of the machine learning algorithms, especially algorithms already created such as those in scikit-learn[7], usually need Standardization, what means to transform the data to looks like standard normally distributed data (Gaussian with zero mean and unit variance). The easiest way to achieve this is to remove the mean to each feature and then scale it by dividing non-constant features by their standard deviation.

Finally, instead of standardize the data, we can scale it by dividing by a constant to make them lie between a minimum and a maximum (usually 0 and 1 respectively). In a RGB image with 8-bit depth, every pixel value will be scaled by 255. The motivation to use this scaling is to increase the model's robustness to small standard deviations while preserving zeros in sparse data.

2.2. Machine Learning

In the last decades the amount of information that is available for processing has not stopped growing, so a large amount of algorithms have been designed and implemented to analyze this information. This is one of the main reasons of the great success of Machine Learning (ML) methods. ML is a field of artificial intelligence that designs programs that are able to learn from an input, what is to say that instead of explicitly programming a computer program to solve a particular problem ML designs a generic algorithm that learns from experience. The programs can manage massive amounts of data by learning patters from them without human interaction. It combines aspects from Statistics, Probability and Algorithmic theory.

There are three basic kinds of Machine Learning:

- Supervised Learning: The Algorithm learns from a training set that is already labeled. Majority of Machine Learning algorithms use supervised Learning. There are another possible division:
  - Classification: When the input has to be classified into classes, the algorithm learns to which class should belong each input. This project will focus on this kind of algorithms.
  - Regression: The output is continuous.
- Unsupervised Learning: It is applied when we do not have labeled examples, the program tries to discern interesting patterns in the input data.
- Reinforcement Learning: In this approach, the program interacts with an environment to realize a specific task, and it will be training continuously using a trial and error approach.

2.2.1. Supervised Learning: Classification

In this kind of problems we build models capable of classifying the data into 2 or more classes (binary classification). All algorithms of this ML methods are structured in the same way.

Manage the Data

We start dividing our data in 2 different sets:

- Training Set: That will be the data that is previously classified before training. so this data will be used to learn the model.
- Test Set: We will use this set to evaluate the model. Using this data we will check the model accuracy, and other statistical measures.

Finally, we can split our dataset into a third kind of set. It is called validation set, and it will be a subset of the training set. It is used for cross-validation, which consist on validating the data while learning, allowing us to generalize the model and prevent problems such as overfitting. Cross-validation technique is used to validate the quality of the model several times with different subsets of the training set called validation sets.

Training and test the model

Once the dataset have been splitted, we can start training the model. During the training the cross-validation technique can be used. The model will be trained until the

stop condition (a preset error estimation, a maximum number of iteration...) is met. Finally the model will be tested with the test set.

There are two main reasons why our model couldn't be able to classify as good as it could be expected:

- Under fitting: The model is not able to fit the training data, so it will get bad results fitting the training set and the test set. This could happens, among other reasons, when the training set does not represent well the dataset to classify, and it usually appears when the training set is too small.

- Over fitting: The model fits perfectly the training data, but it can not generalize to new samples, so it will obtain bad results with the test set. This problem usually appears when the model learns too much details, or learns also the noise in the training set.

A good model is the one which can avoid these problems, learning enough from the training set and generalizing later. We must have these problems in mind when building a model and, although there are some techniques to prevent them, it is very difficult to achieve a model that completely avoid overfitting and underfitting issues.

2.3. Support Vector Machines

Support Vector Machines are one of the most popular supervised learning's algorithms, developed in the lately 1990s to use in some problems of time series prediction, face recognition and medical analysis. Their success are based on their strong mathematical base, encouraging researchers to go deeper and looking for new applications.

If we have L training Points, and each input x_i has D attributes and the goal is to obtain a binary classification, what is to say that we can build each input as a tuple (x_i, y_i) where y is 1 or -1, and $x_i \in R^D$. Our first assumption will be that our data is linearly separable. Then if D=2 we can separate the 2 classes by a line, or by a hyperplane if D>2.

So build a SVM is the same than finding the best hyperplane for the input data (the hyperplane that better divides all of the input, or instances, into 2 classes). It can be described as $w \cdot x + b = 0$, where w is normal to the hyperplane and $\frac{b}{\|w\|}$ is the perpendicular distance from the hyperplane to the origin.

As we can see in Figure 2.3 the best hyperplane is the one which maximizes the distance between the closest members of both classes, in this figure d_1 and d_2 which are the distances from H_1 and H_2 respectively to the hyperplane. Now we can describe our data by:

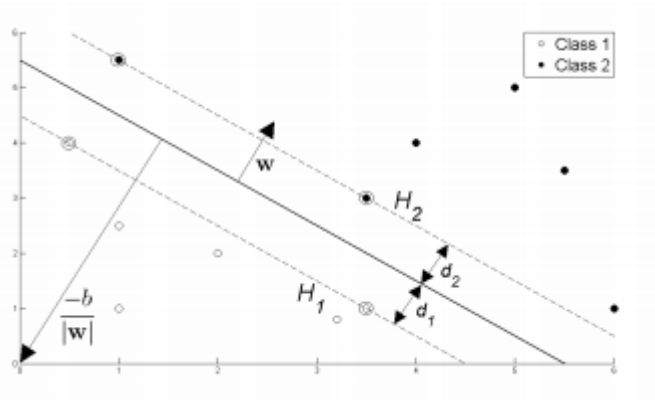

Figura 2.3: Hyperplane construction when D=2

$$x_i \cdot w + b \geq +1, \quad \text{for} \quad y_i = +1 \tag{2.1}$$
$$x_i \cdot w + b \leq -1, \quad \text{for} \quad y_i = -1 \tag{2.2}$$
$$\tag{2.3}$$

That equation is the same that

$$y_i(x_i \cdot w + b) - 1 \geq 0, \quad \forall i \tag{2.4}$$

Being equality the case of the points that are closer to the hyperplane. So applying basic geometry we can see that maximizing that margin is the same as $\frac{1}{\|w\|}$, so we need to look for min $\|w\|$, what is the same than minimizing $\frac{1}{2}\|w\|^2$, which give us the condition

$$\min \frac{1}{2}\|w\|^2 \quad \text{s.t} \quad y_i(x_i \cdot w + b) - 1 \geq 0 \tag{2.5}$$

Now, to look for w and b, we need to use Lagrange Multipliers $\alpha \quad \text{s.t} \quad \alpha_i \geq 0$:

$$L_P \equiv \frac{1}{2}\|w\|^2 - \alpha[y_i(x_i \cdot w + b) - 1, \forall i] \equiv \frac{1}{2}\|w\|^2 - \sum_1^L \alpha_i y_i(x_i \cdot w + b) + \sum_1^L \alpha_i \tag{2.6}$$

We want to find w and b that minimizes L_P, so we can differentiate it with respect to w and b, and then equal both to 0. So we get a new equality for w and another one for b, then we substitute it into 2.6, to obtain:

$$L_D \equiv \sum_{i=1}^{L} \alpha_i - \frac{1}{2}\alpha^T H \alpha \quad \text{s.t} \quad \alpha_i \geq 0, \forall i, \quad \sum_{i=1}^{L} \alpha_i y_i, \quad H_{ij} = y_i y_j x_i x_j \tag{2.7}$$

Now instead of looking for minimizing L_P we are trying to maximize L_D, which depends on α, so we are looking to maximize 2.7. So this is a quadratic optimization problem that can be solved using Quadratic Programming, which will return α and hence, w.

Finally we only need to find b. Any data point satisfying $\frac{\partial L_P}{\partial b} = 0$ which is a support vector x_s that will have the form:

$$y_s(x_s \cdot w + b) = 1 \tag{2.8}$$

and hence, where S denotes the set of indices of the support vectors:

$$y_s(\sum_{m \in S} \alpha_m y_m x_m \cdot x_s + b) = 1 \tag{2.9}$$

it follows that

$$b = y_s - \sum_{m \in S} \alpha_m y_m x_m \cdot x_s \tag{2.10}$$

This is the basic functioning of a Support Vector Machine, however, we have supposed that the input data was linearly separable, and it is not always possible (in fact, almost never is). So now we are going to see how a support vector machine works with data that is not linearly separable.

If we return to 2.7 we can see that we created a matrix H from the dot product of our input variables:

$$H_{ij} = y_i y_j k(x_i, x_j) = x_i \cdot x_j = x_i^T x_j \tag{2.11}$$

This function k(x_i,x_j) = $x_i^T x_j$ is called a kernel (this one is the linear kernel). All kernels are based on calculating the inner product of 2 vectors, so we can us them to recast features into a higher dimensionality space by a non-linear mapping (non-linear kernel) as we can see.

For example, with the Radial Basis Kernel, which is defined by:

$$k(x_i, x_j) = e^{-(\frac{\|x_i - x_j\|^2}{2\theta^2})} \tag{2.12}$$

we can recast some non linearly separable data into a higher space where it is separable, it is shown in Figure 2.4:

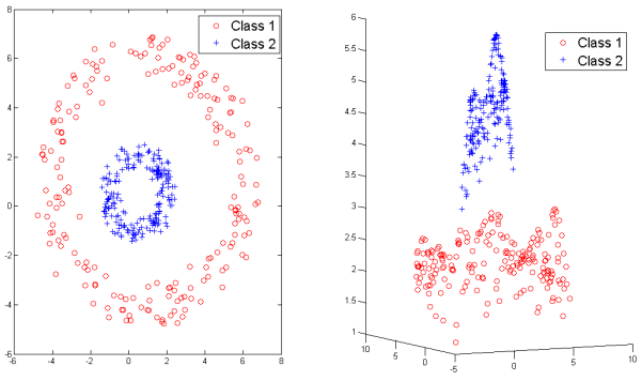

Figura 2.4: 2D Feature Mapping with Radial Basis Kernel

It is easy to see that after mapping the data has became separable. Another important kernels are:

- Polynomial kernel: $k(x_i, x_j) = (x_i \cdot x_j + a)^b$
- Sigmoidal Kernel: $k(x_i, x_j) = \tanh(ax_i \cdot x_j - b)$

where the parameters a and b define the behavior of the kernel[8] .

2.4. Artificial Neural Networks

Artificial Neural Networks (ANN) are computing systems that are based on biological neural networks. The first learning hypothesis about systems based on biological neural networks was presented on the 1940s, and there were slow advances in this area during decades. In 1965 researchers discovered that there were several problems to advance in this area, including the lack of computational capability. There were little advances till the 21 century, when that requisites started to be fulfilled, specially with the development of GPU based computation[9].

First we will see how the biological neurons work. As we can see in Figure (2.5) any biological neuron is formed by a nucleus, which is connected to other neuron's nucleus by the dendrites (input) and the axon (output), forming a connection called synaptic connection. The neurons can fire electric pulses through this connection, receiving the pulse by the dendrites, and when it has received enough information it propagates a signal through the axon to other connected neurons. This is how the information propagates through the neural network. Both the synaptic connection and the pulses needed to activate a neuron change during its lifetime, which is what allows the network learn. We can not implement an artificial brain due to it's high complexity, but we can make simpler networks formed by artificial neurons which are able to recognize patterns and describe complex problems with simple rules, as it has been demonstrated in the literature. This artificial neural networks have a high learning capacity and they are good at generalizing from a training set[10].

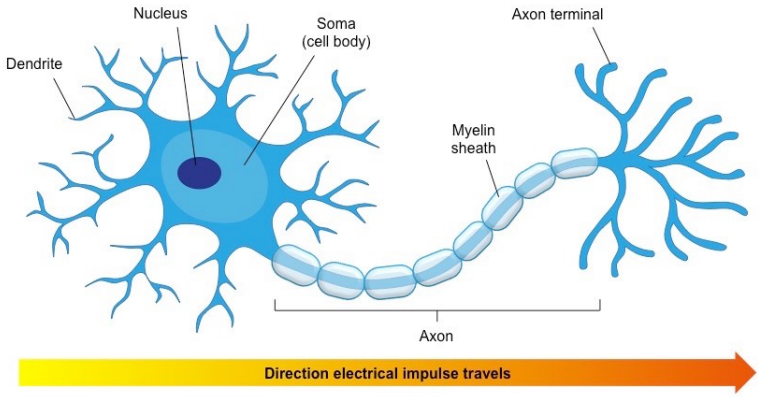

Figura 2.5: Biological Neuron

A standard neural network (NN) consists on many simple, connected processors called neurons, each producing a sequence of real-valued activations. Inputs neurons are activated by weighted connections with other neuron's inputs, or through sensors which perceive the environment

The general mathematics definition of an artificial neuron is

$$y(x) = \varphi(\sum_{i=0}^{n} w_i x_i + b) \tag{2.13}$$

being y(x) the output axon, and $x_0, ..., x_n$ are the n input dendrites, which are weighted by $w_0, ..., w_n$, that in the biological network would be the number of pulses needed to activate the function. Sometimes, a bias term b is added to this weighted sum to serve as a threshold for the activation function. Finally, φ is the activation function which determines the powerful that outputs are. The function g is preferable to be differentiable and it usually returns a result between 0 and 1 or between -1 and 1, although there are some activation functions without these constraints[11]. A simple Artificial Neuron structure is shown in figure (2.6).

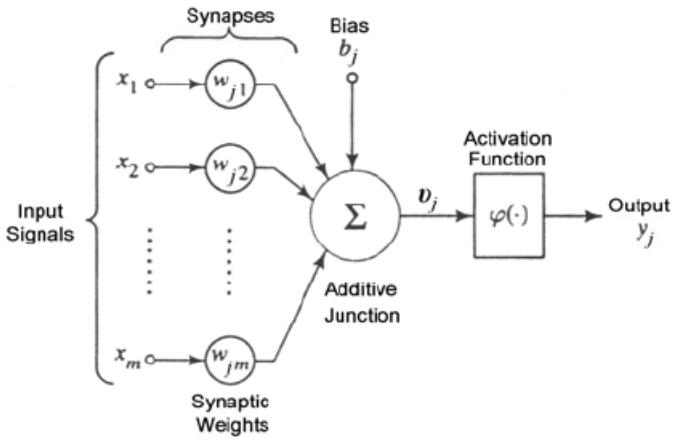

Figura 2.6: Artificial Neuron

Any Artificial Neural Network (ANN) is formed by layers, which are a collection of neurons operating together at a specific depth. There are 3 types of layers: input layers, which receive the raw data; output layers, which is the last layer of the network; and the hidden layers, which are the layers between input and output ones. The data enters through the network by the input layer and propagates to the last layer, generating the output. Then the ANN computes the error term and uses the **Backpropagation** Algorithm, which modifies the weights by dividing the error equally between each neuron.

Backpropagation Algorithm

The Backpropagation algorithm has been used in different contexts, and it has been the most studied and used algorithm for neural networks[12]. This algorithm uses the method of gradient descent looking for minimizing the error function in the weight space. The network gets a list of pairs $(x_1, t_1), ..., (x_p, t_p)$, then, each input pattern x_i produces an output o_i which is usually different from the target t_i (the algorithm main goal is to get an equal o_i and t_i). To be precise, we want to minimize the error function:

$$E = \frac{1}{2}\sum_{1}^{p} \|o_i - t_i\|^2 \tag{2.14}$$

To minimize this error first we will initialize the weights randomly, then the gradient of the error is computed and the initial weights are corrected. So our task is to get the gradient error recursively.

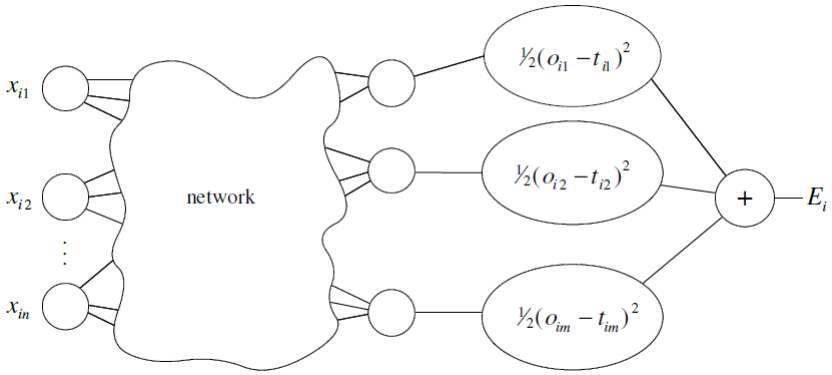

Figura 2.7: Extended network for the error function's computation

First we will extend the network as shown in Figure (2.7), where each j-output is connected to a node that evaluates the function

$$E = \frac{1}{2}(o_{ij} - t_{ij}) \tag{2.15}$$

where j denotes the j-th component of the vectors o_i and t_i. The outputs of the additional m nodes are collected at a node which adds them up, giving as output E_i. Then we will repeat the process for each one of the patterns t_i, getting at least $E_1, ...E_p$, whose sum is the error function E. Now that we have a way to compute the total error of the training set, and the only way to minimize it is to modify the network weights. Then, as E depends only from the weights $w_1, ..., w_l$, thus we can calculate the gradient as follows:

$$\nabla E = (\frac{\delta E}{\delta w_1}, ..., \frac{\delta E}{w_l}) \tag{2.16}$$

and then update each weight using the gradient as equation 2.17 shows:

$$\Delta w_i = -\gamma \frac{\delta E}{\delta w_i} \text{for i} = 1,...,\text{l} \tag{2.17}$$

This way the problem is reduced to compute the gradient of the network function w.r.t its weights. Once a method for this purpose is implemented, we will adjust the weights with it iteratively, expecting to find a minimum of the error function where $\nabla E = 0$[13].

2.5. Deep Learning

As stated before, the Machine Learning algorithms are those capable of extracting patterns from raw data to acquire their own knowledge. However, these algorithms depends always on the way we have to represent the data, i.e, it's not the same to use a binary classification model to classify patients between healthy or not if the input data is a raw X-Ray Image or a set of features extracted by the doctor from the same image. Depending on how the data is represented the result can be very different.

This approach leads us to the task of finding a new algorithm which not only knows how to find patters, but also finds the best way to represent the data. This approach is called **Representation Learning**, and they get a better performance in representation tasks than those hand-designed. In addition, it works much faster (from hours to months, depending on the problem complexity) in tasks that could take several years if they are done manually (or even it couldn't be processed by humans due to the high complexity or volume of data). However, the complexity of the representation problem can scale until the point where finding a good representation it is as difficult to solve than the original machine learning problem itself. Is in those cases where the representation learning can not help us.

Deep Learning models are large and deep neural nets with more hidden layers and more nodes solves. These models have been recently implemented thanks to the development of much more powerful computers. They solve this problem by representing the data in terms of simpler representations, allowing the systems to build complex concepts from simpler concepts. Another perspective on deep learning is that depth enables to learn a multi-step computer program. A network with greater depths allows to execute more instructions in sequence, allowing each set of instructions to use the output of previous executions as input.[14]

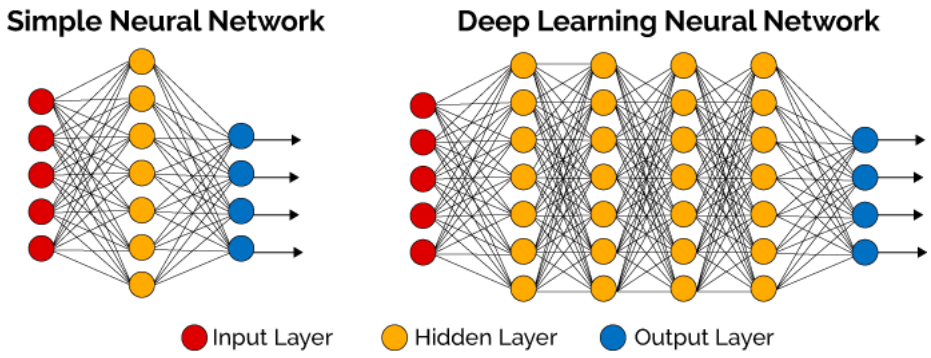

Figura 2.8: Simple Neural Network vs Deep Neural Network

2.5.1. Transfer Learning

Many Machine Learning algorithms work well under one assumption: the training and test data are drawn from the same feature space and distribution. However, in many cases, if this distribution changes a new model needs to be created but is not possible to rebuilt these models with the new re-collected data. In those cases in when transfer learning between task domains would be desirable[15].

Precisely, when training an ANN is when Transfer Learning is needed, because to train an ANN a very large dataset is needed to get optimal results. Finding a method which can use knowledge from another model (for example one where a large dataset is easily found) would decrease the required size of the dataset to train the new model. In particular there is a Transfer Learning technique widely used in image recognition: Fine-tunning[16].

Fine-Tunning

Based on one assumption: that the features extracted in first layers of the ANN are more general, and those extracted in deep layers are more specific. We procces using the previous strategy, removing the last fully-connected layer of the model, and retraining the ANN. Then we will add a new fully-connected layer which output will be an array of length the number of classes. Then we can retrain only the last layer, or a set of layers. We can even train the full ANN again, with the difference that the initial weights will be those of the pre-trained model.

2.6. Convolutional Neural Networks

We are interested in Convolutional Neural Networks (CNN), which are a class of feed-forward artificial neural networks, and are mainly used for image recognition. They are also based on biological processes, the connectivity pattern of neurons that form the animal visual cortex.

The basic architecture of a CNN is the same as a Regular NN: it is formed by neurons with learnable weights and biases, and when they receive an input, they perform a dot product and an optional non linear operation. This, will a loss function and a fully connected layer will transform the raw data into a class score at the end. CNNs are characterized by having at least one Convolutional Layer.

2.6.1. CNN Layers

Convolutional Layer

Due to the high-dimension of the images, connecting each neuron to every neuron in the previous volume is not the best solution. Instead, each neuron will be connected only to a local region of the input volume. A convolutional layer consists on a set of filters which is small spatially, in terms of width and height, but extends through the full depth of the input, what is to say the channels. This spatial extent is a hyper parameter called Receptive Field (RF). We will apply every filter across the width and height of the input and we a 2D matrix called activation map will be built, it is the response of the filter for each spatial position. A convolutional layer output is built from stacking all activation maps produced for every filter.

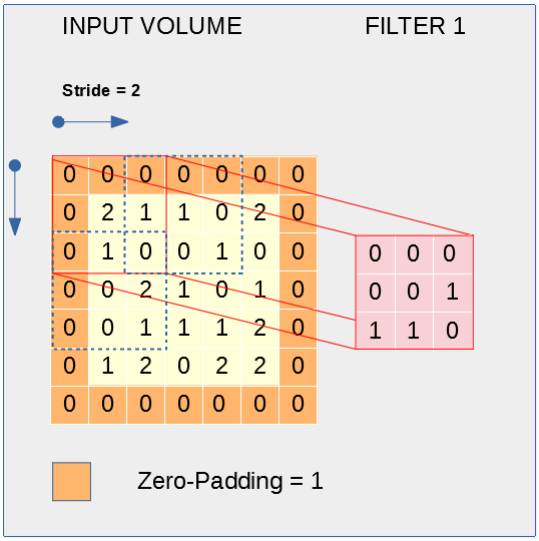

Figura 2.9: Convolutional Filter

We have explained how connect neurons with the input, now we are going to describe how to connect them to the output. The size of the output is controlled by 3 hyperparameters:

- Depth(D): It is the number of filters, each one looks for something different in the input space, and the set of neurons that are all looking in the same spatial region (local region of the input which is connected to that neurons) of the input as depth column.
- Stride(S): It is the number of pixels that we shift the filter across the width and height of the input. The bigger the stride, the smaller the output.
- Zero-padding(ZP): We can add a border of zeros to the input volume to control the size of the output. The size of this border is called zero-padding.

Then, if we call IS to the input size, we can compute the spatial size K, that represents the width and height of the output as:

$$\frac{IS - RF + 2ZP}{S} + 1 = K \tag{2.18}$$

So, the output volume will be of size [KxKxN], which means that there are $K*K*D = N$ neurons in the convolutional layer. Now, supposing no zero-padding for simplicity, each one of the N neurons will be connected to a region of size [RFxRFxD]. Then each one of the N neurons has $RF*RF*D$ weights plus the bias, which gives $K*K*N*RF*RF*D$. This will be the number of parameters the convolutional layer has.

For controlling this number of parameters, a parameter sharing scheme is used in Convolutional Layers, and it is based in one assumption: that if one feature is useful at some spatial position(x_1,y_1), then it is also useful at other position (x_2,y_2). Then, taking the output of the convolutional layer of size [KxKxN], we will denote N as depth slices, and the slices will have a size of [KxK]. Therefore we will use the same weights plus bias for every each slice, which will give us N sets of unique weights (plus bias). So the final number of parameters of the convolutional layer will be $RF * RF * D * N$.

Note that this assumption is not always true, specially with centered structures in the input images, where we can find completely different features in one side of the image. In this case we have to relax the parameter sharing scheme.

Finally, we have to take into account some considerations: the backpropagation over the convolutional layer is also a convolution, and there is another kind of hyperparameter which was introduced later, and it is called *dilation*, which is based on the idea that the filters do not have to be continuous, i .e, can be a number of cells (dilation) between each filter application. It is useful to merge spatial information across the input in a much more aggresive way.

Pooling Layer

This layer is used to decrease the spatial size of the data, and it is usually added between convolutional layers. It takes 2 hyperparameters: Spatial Extent (E) and Stride(S). The layer implements a filter of size [ExE], applying an operation and shifting S pixels after it. There are several kind of pooling:

- max pooling: It takes the maximum value of the $E * E$ possible values
- average pooling: It takes the average value of the values
- L2-norm pooling: compute the L2-norm of the values.

This helps to decrease number of parameters and computation requirements and hence, to prevent overfitting.

Fully Connected Layer

Fully-Connected Layers(FC) have full connections to all activations in previous layers. Like in convolutional layer, FC neurons also perform a dot product, so their functionality is identical, but convolutional layers are connected only to a region of the input (output of the previous layer), and FC are connected to all the input.

2.6.2. Pre-Trained Models

In this job we are going to use 4 different pre-trained models (4 different CNN), applying Transfer Learning to train them. We will use models that already have a name and we will load the weights got by training the model with the imageNet dataset.

VGG16 and 19 Neural Networks

It was introduced by Simonyan and Zisserman in their 2014 paper, *'Very Deep Convolutional Networks for Large Scale Image Recognition'*. It has a very simple architecture.

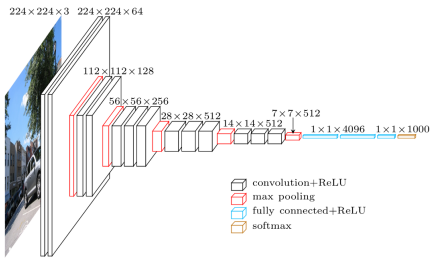

Figura 2.10: VGG architecture

It is formed by 3x3 convolutional layers stacked in top of each other in increasing depth, and reducing volume size using max pooling. In the end there are 2 fully-connected layers, each one with 4096 nodes, and a softmax classifier. The difference between VGG16 and VGG19 is that the first one has 16 weight layers and the second one has 19 (columns D and E in the figure 2.10).

VGG networks were developed in smaller versions first (columns A and C in 2.11). Smaller versions converge and are used as initializations for the larger ones, in a process process called pre-training. However, this pre-training is quite expensive in terms of time. For this reasons this initialization was changed by Xaiver/Glorot initialization or MSRA initialization, which consists on initializing the weights following a Gaussian distribution. Even with that, there are 2 main problems in VGG nets: it is very slow to train and the weights are also very large in terms of disk/bandwidth.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figura 2.11: Table 1 of Very Deep Convolutional Networks for Large Scale Image Recognition, Simonyan and Zisserman (2014)

InceptionV3

The 'Inception' micro-architecture was first introduced by Szegedy et al. in their 2014 paper, *'Going Deeper with Convolutions'*. The original one (Inception V1) was also called googleNet.

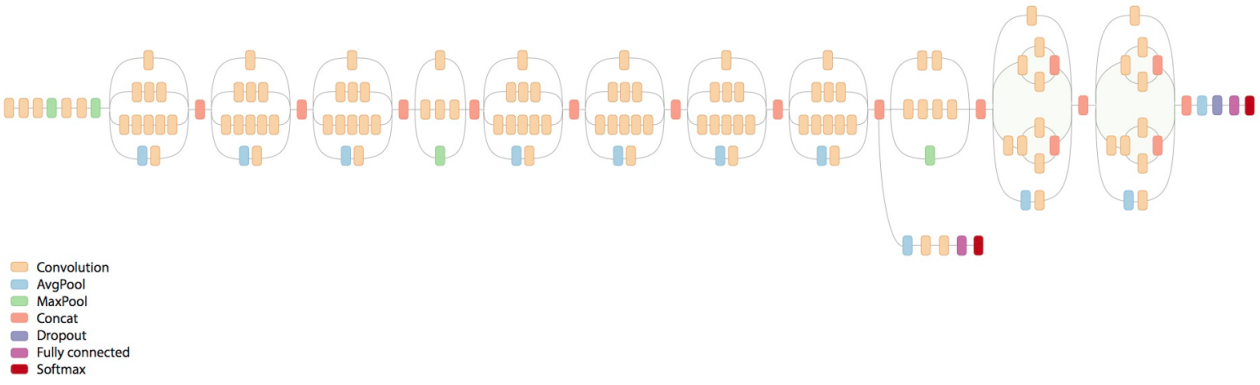

Figura 2.12: Inception V3 architecture

It is based in a multi-factor feature extraction with 1x1, 3x3 and 5x5 convolutions, whose outputs are stacked along the channel dimension and it feed the next layer in the network. This inception architecture that we are going to use in this work is the one implemented in keras, which comes from the later publication by Szegedy et al., Rethinking the Inception Architecture for Computer Vision (2015), which works better with imageNet weights.

resNet50

First introduced by He et al. in their 2015 paper, *'Deep Residual Learning for Image Recognition'*, its architecture could be described as a network of networks, because is formed by blocks (each one with their convoluton, pooling...), which leads to the final net. An advantage of resNet50 is that is much deeper than VGG nets, but its size is much smaller due to the use of average pooling.

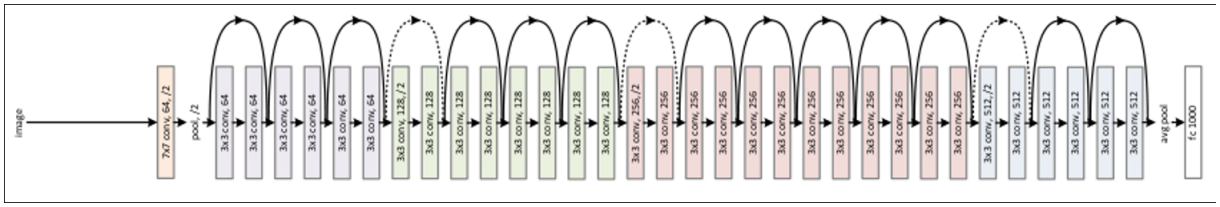

Figura 2.13: resNet50 architecture

3

Machine Learning Methods for Medical Image Classification

To analyze the performance of the ML algorithms explained in section (2) we will build a model which classifies X-Ray Images from patients with tuberculosis (TB) and from Healthy Patients. To achieve this, different processing techniques will be applied to a dataset of X Ray images of similar characteristics than those used in medical projects. Then several SVM and CNN models will be trained and analyzed. Finally the best models will be chosen to build an ensemble.

3.1. DataSet

The dataset chosen for the project is the one published in *'Two public chest X-ray datasets for computer-aided screening of pulmonary diseases'*[17], and it's formed by two different datasets. The first one is formed by 138 frontal chest X-rays from Montgomery County's Tuberculosis screening program, being 80 of them normal images and the other 58 from patients with TB (abnormal from now on). All of them have size either 4020x4892 or 4892x4020 pixels. The second dataset has 662 frontal chest X-rays, of which 326 are normal cases and 336 are abnormal cases, and their size vary, being close to 3000x3000 pixels.

3.2. Pre Processing Data

As we mentioned in the last section, the main problem for deep learning algorithms over medical images is the small number of images available. For this reason the first step is to enlarge the size of the dataSet. This process is called Data Augmentation.

We will create two different folders, the first one for normal images and the second one for abnormal images. Then we will resize every image to 224x224 pixels and we will move it to the corresponding folder. After this, we will split 90 % of the images for training the models and the other 10 % for testing them. We will split the images before shuffling them to train and test each model with exactly the same images, this way, it will be easier to compare each model's performance with different processing methods.

Then we will start with the methods explained in section (2). First of all we will duplicate every training image by mirroring them, as shown in figure 3.1-(b), improving the models generalization by allowing them to identify similar features in the opposite side of the thorax than those found in the original training image. This is the first method because we want to apply the same processing methods to mirrored images than to the original ones.

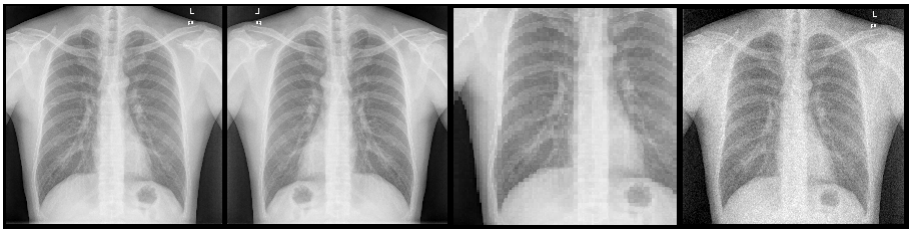

Figura 3.1: Image Processing Based on Positional Approach. From left to right: (a)Original Image, (b)Mirror Image, (c)Cropped Image, (d)Image With Gaussian Random Noise

Now we will apply one of the Histogram Equalization methods explained before, trying to find which algorithm (and it's parameters, as explained in section (2)) fits best this dataSet. So we are going to create nine new datasets, each one of them with the images got from mirroring (and the original ones too) after applying the Histogram Equalization or the CLAHE method with clip Limit = i and Tile Grid Size = $(2^i, 2^i)$ and $(2^{i+1}, 2^{i+1})$, for $i = 1, 2, 3, 4$.

As we can see in figure 3.2, these methods enhance the image contrasts, and if we look at the ribs we can easily see the difference between both types of histogram methods. While the normal histogram equalization enhances the general contrast, making harder to differentiate the ribs form the background of the image, and the CLAHE methods enhances the contrast in small areas, increasing the ribs visibility.

Then, we will add noise to every image with a probability of 25 %, 50 % or 75 %. The noise have the goal of improving the model's robustness, allowing the model to avoid making a misclassification by an outlier or a distortion in the image, that can happen for many reasons, i.e, due to an interference in the machine which takes the X-Ray Images. However, we will add the noise only to the training set.

Figura 3.2: Image Processing Based on Histogram. From left to right: Original Image, Image after applying histogram equalization, Image after applying CLAHE with clip Limit = 2 and Tile Grid Size = (4,4)

The last method, random cropping, will be used only for training the CNNs, because the SVM does not allow it. As explained in section (2), the SVM represents each image with dimensions *nxn* as a point in the (n^2)-dimensional plane, and looks for the hyperplane of n^2-1 dimensions which separates the best possible the samples in two classes, something that would be impossible if each sample would have different dimensions.

To perform the random cropping we will use the data structures existing in the Keras Library[18], the one chosen for building the CNNs. Before starting to train the CNN models we have to create a structure of type *ImageDataGenerator* that will contain the processed images and will represent it as the models require, allowing the user to automatically perform some operations to the data. By using this data structure, we will add a random cropping with zoom range of 99 pixels, which means that the image resulting for it will have a size of 198x198 pixels. This operation is performed by the library, applying the random cropping sometimes, and working with the full image other times.

Taking advantage of this data-structure we will also scale the data that will be used to train and test the CNNs by a factor of 1/255, which avoids the weights in the model to grow too high and increasing the model robustness against small standard deviations.

To the dataset that will be used to build the SVM models, instead of scaling the data, we will standardize it, which will be done by removing the mean and scaling by the standard deviation of the image.

The figure (3.3) shows the full path that follow the images since they are downloaded until they are prepared for training or testing the models.

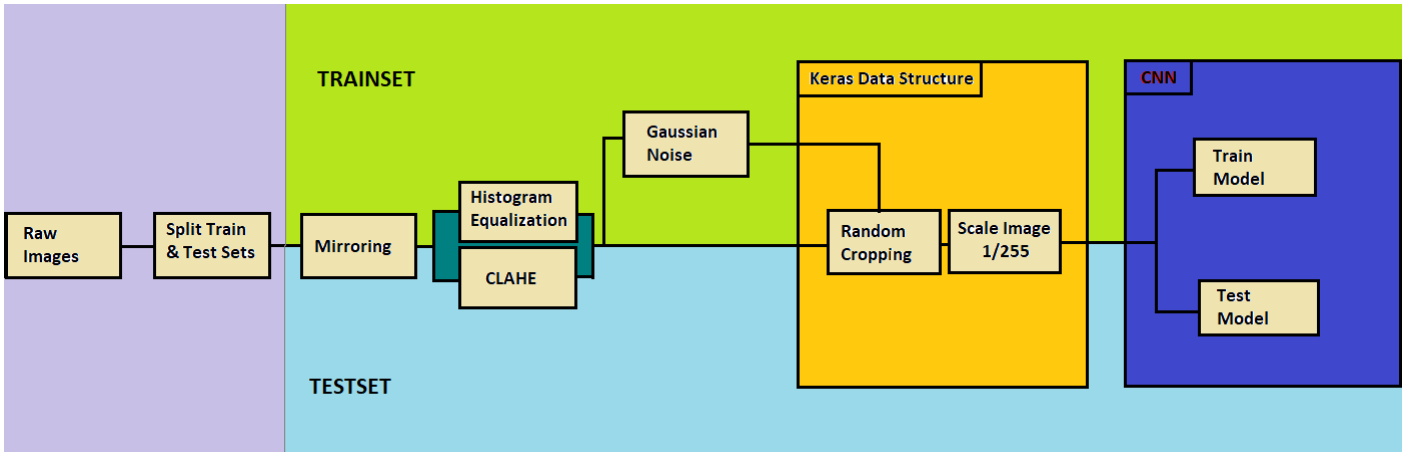

Figura 3.3: Data Flow Diagram

3.3. Training The Models

Once we have our data preprocessed, we will start training the models. To do this we will train the five models (the four neural networks and the SVM) with different parameters.

CNN Methodology

For the CNN models we will use the pre-built models existing in the Keras library for the ResNet50, InceptionV3, VGG16 and VGG19, but as explained in section (2), the images available for training the model, even after data augmentation, are not enough for training a good CNN model, so we will use the fine-tune technique.

For this reason we will load the 'imagenet' weights, which were trained during the 'ImageNet Large Scale Visual Recognition Challenge'[19], where the models were trained with nearly 1.2 millions images, another 50.000 images for validation and 100.000 for testing. During this competition, the models were trained for object detection and object classification, but we are interested only in the second one, achieving a mean accuracy in this task higher than 90 %.

Once the weights are loaded, we will remove the last layer of each CNN, which is the classification layer, and we will add a new classification layer which will classify the previous layer's output into 2 different classes. The first parameter to be modified to look for the best possible model will be the activation functions of the classification layer. We will train the models with the following activation functions: linear, sigmoid and softmax.

Then, before start training the model, it has to be compiled with the binary cross entropy function, which computes the loss value for the backpropagation algorithm as $(ylog(p) + (1y)log(1p))$. To compile the model we also have to choose the optimizer (choosing between the sigmoid, Adam or Nadam optimizers), and the learning rate.

Finally, we need to prepare the preprocessed data for training the model. To do this we will create, as explained before, an *ImageDataGenerator* data structure to scale the data and select the random cropping zoom range. In addition, we will set the validation split as a 10 % of the training samples. Then, we will start training the model. Because of how long the algorithm takes in every training iteration (epoch), we will save the model each 10 epochs with the keras library function, which saves the weights and the compilation status. For this reason, to train a 30-epoch model we will load the 20-epoch model and we will train it for 10 more epochs. This will allow us to stop training the models and resume it later, or to train a model again in much less time that it would take to train it from the beginning.

Our first try will be to freeze all layers except the last one (the classifier). Frozen layers won't be trained any more, what is to say that the weights won't be modified. This way we will use the CNN's pretrained weights as a feature extractor, allowing us to get good results by training the models only for a few epoch (between 1 and 10).

After this, we will build new models without freezing any layer, also modifying the parameters explained before (activation function of the classification layer, the optimizer and it's learning rate). The models will be trained for 20 to 50 epochs.

Finally, we will test every model with the testing samples and we will choose the best parameter's configuration. We can see all of this in figure (3.4).

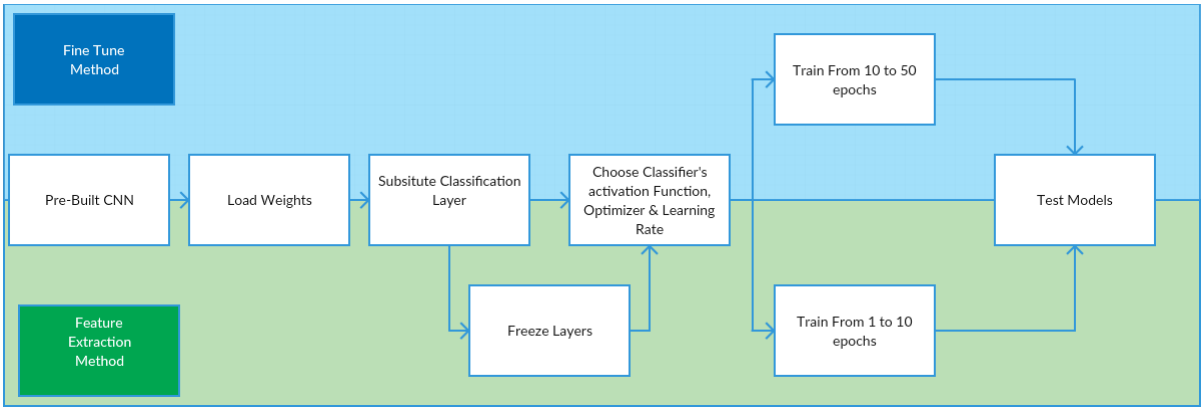

Figura 3.4: Model's Configuration Schema

SVM Methodology

To prepare the pre-processed images to train the SVM models, we will standardize every image in the augmented dataset, and then we will flatten them from a matrix of shape 224x224 to a 1D array of 1x50176. Then we will build different models changing it's kernel. The kernels used will be, as stated in section (2) the linear, sigmoid, rbf and the polynomial kernel with degrees from 2 to 8. Then we will test the models to find the kernel which fit better with this dataset.

Finally, we will use the SVM models to classify the features extracted by the CNNs. To do this we will remove, as explained in CNN methodology, the last layer of each Neural

Network (the classification layer), which will give us an array of dimension 1x4096 in the VGG16 and VGG19 models, what is to say the CNN will extract 4096 features of each image, and they will be used to classify the image using the SVM. On the other hand the resNet50 and the InceptionV3 model will return an array of 2048 features each one. With this features (that are already flattened), we will train other SVM models.

Ensembles

Once we have trained and tested the models, we will analyze their results. To do this we will set a criterion to measure each model's performance, because focusing only on the overall accuracy of the model could lead us to choose a worse model than if we look at other criterion. This criterion is explained in section (4).

Then we will choose the 3 algorithms which achieve the best results according to the criterion chosen. We will not use the same algorithm with different parameters, i.e, we wont use the VGG19 model with different activation function and/or optimizer. This is because we want to improve our stand-alone models assuming that there are higher probabilities of classifying correctly an image if it has been classified by two different models. However, if (at least) two of the models that make up an ensemble classify each image the same way, what is to say that if they extract the same features then the ensemble will classify each image the same way than those stand-alone models.

We will build two different ensembles, applying to their results the same performance criterion than to the stand-alone models. Both ensembles will receive an array with the predictions made by the stand-alone models for the testing data, that will be obtained with the function *predict(testing data)* for both the CNN and the SVM models. This array will be formed by a list of two numbers between zero and one for each input image, that will correspond to the probability of the image to belong to the class 0 (Normal) or 1 (Abnormal) according to that stand-alone model.

Then, the ensembles will be:

- Ensemble 1: The first ensemble will not take into account the probability of each image belonging to each class. Then, the ensemble will classify each image as it has been classified by the maximum number of stand-alone models, what is to say the image will be classified as zero if it has been classified as zero by 2 (or more) of the stand-alone models that form the ensemble, and the same happens to classify an image as one.

- Ensemble 2: The second ensemble will not necessarily classify an image as it has been classified by 2 of the stand-alone models. Instead of that, it will get the probability of an image to belong to a class as the sum of the probabilities given by the stand-alone models. The result of this sum will be between 0 and 3, so if an image gets a result higher than 1.5 of belonging to a class, then the image will be classified as that class. This is, in fact, the average probability of belonging to each class.

Finally, we will analyze all results, looking for the best trained model, and we will discuss the future work needed to use our model in a real diagnosis service.

4 Results

First let's define how we are going to measure each model's performance. Then we will analyze each model's results, evaluating them and choosing the three best models for our ensembles.

4.1. Performance Measurement Criterion

Before analyzing the results of each model, we must establish a criterion to study the performance of the models. As explained in section (1), the goal of these machine learning models is to remove or decrease the use of other tests that are longer, more expensive, invasive or dangerous. For this reason, if our criterion set the best model as the one with the highest total accuracy, it could lead to choose a model which will have worse performance than others.

if one of these models were used to diagnose, it should serve to decrease the number of tests that are done to patients who are candidates for tuberculosis. But in medicine, an error can be dangerous, which can lead to patient's death, so if the diagnosis is going to be based in the model's results, a high statistical evidence must be given by the model. For this reason we will divide misclassifications between patients with tuberculosis which are classified as healthy (False Negative or Type I Error) and healthy patients classified as patients with tuberculosis (False Positive or Type II Error). We will look for models which during the test didn't give false positives or false negatives.

Then, if our goal is to get the model which give us the highest percentage of correct classifications, with an error probability that we can considerate despicable, that in our case, it has to be as close as zero as we can get. For this reason, we will look for a

model without false positives, which means that if it predicts that the image belongs to a tuberculosis patient then the prediction is almost sure correct, or without false negatives, that means that if the prediction is healthy, then it is almost sure correct.

In medicine, a treatment can be only be given to a patient if there are a high evidence that is the correct one, and an automatic classification model could not be accepted as a evidence high enough even if the false positives or false negatives are zero in every test made, because the small number of samples that use to be available for this kind of images. For this reason, we are more interested in a model without false negatives, because it can be used to avoid further tests in patients classified as healthy.

4.2. Models Results

4.2.1. Stand-Alone Models

SVM

In the SVM models, we used 90 % of the data to train the model, and and the other 10 % to test it. After testing it I realized that with the Standard Histogram Equalization gives better results, however, the sigmoid and the rbf kernel classify every image as the same class, which means that this kernels are not suitable for this dataset. We can see the results in table (4.1):

Kernel	Linear	Pol. Deg. 2	Pol. Deg. 3
Standard Histogram Equalization	**0.5802**	**0.6296**	**0.642**
CLAHE	**0.6049**	**0.617**	**0.617**

Tabla 4.1: SVM best accuracy result for each kernel

Analyzing these results we see that the SVM model with a polynomial kernel has the best accuracy, and it increases as long as the degree of the kernel does, so we are going to increase that degree until we find it's limit. The table (4.2) shows us the result of the SVM models with polynomial kernels for degrees from 2 to 8:

Degree	2	3	4	5	6	7	8
Accuracy	**0.6296**	**0.642**	**0.6543**	**0.6543**	**0.6543**	**0.679**	**0.666**

Tabla 4.2: SVM With polynomial kernel of degree 2 to 8

As we can see, the SVM accuracy increases as the degree of the polynomial kernel increases, getting his maximum with degree 7. However, these results are very poor, cause a model that classify between 2 classes with an accuracy of a 65 % is not good enough (at least not for basing a diagnose on it).

CNN

To train the CNN models I also used 90 % to train the model, but being a 10 % of that percentage for a validation split, and the remaining 10 % for testing the model. The CNN results are as follows:

	ResNet50	InceptionV3	VGG19
Epochs	30	10	50
Activation Function	linear	sigmoid	linear
Model Optimizer	Adam	Adam	Adam
Learning Rate	0.0001	0.0001	0.001
Histogram Type	Normal	CLAHE	Normal
Accuracy	**0.8395**	**0.716**	**0.7901**

Tabla 4.3: Best model's configuration

As we can see the CNN models have significant better accuracy than the SVM models. If we only look for the model with the highest overall accuracy, then the best model would be ResNet50 trained for 30 epochs, with an Adam optimizer and with a prediction layer with a linear activation function and a learning rate of 0.0001, reaching an overall accuracy of 83.95 %.

	Epochs	10		20		30		40		50	
Model		Normal	Abnormal	Normal	Abnormal	Normal	Abnormal	Normal	Abnormal	Normal	Abnormal
ResNet50	Normal	13	28	40	1	34	7	41	0	18	23
	Abnormal	2	38	17	23	6	34	20	20	1	39
	Accuracy	0.6296		0.7778		0.8395		0.753		0.7037	
InceptionV3	Normal	41	0	41	0	41	0	41	0	40	1
	Abnormal	23	17	33	7	29	11	34	6	20	20
	Accuracy	0.716		0.5926		0.642		0.5802		0.7407	
VGG19	Normal	41	0	18	023	39	2	37	4	38	3
	Abnormal	28	2	20	20	15	25	14	26	14	26
	Accuracy	0.5308		0.4691		0.7901		0.7778		0.7901	

Tabla 4.4: CNN models confusion matrix

However, as explained before, we are not only interested in the overall accuracy of the model, but also in the misclasified's images. To analyze this, we will compute the confusion matrix of each CNN and SVM model.

Kernel	Linear		Polynomial Deg. 5		Polynomial Deg. 6		Polynomial Deg. 7	
	Normal	Abnormal	Normal	Abnormal	Normal	Abnormal	Normal	Abnormal
Normal	30	11	36	5	37	4	38	03
Abnormal	23	17	23	17	24	16	24	16
Accuracy	0.5802		0.6543		0.6543		0.6543	

Tabla 4.5: SVM models confusion matrix

As we can see in table (4.4) and table (4.5) most of the algorithms results show that these models tend to make a better classification of normal ¡images, having more

difficulties to classify abnormal images. The CNN models tends to classify more images as they are trained for more epochs, but it causes an increase of misclasifications.

Looking at table (4.4), we can see that the model which doesn't have false positives and has less false negatives is ResNet50 trained for 40 epochs. It has a total accuracy of 75.3 %, significantly smaller than the same model trained for 30 epochs, which has an overall accuracy of 83.95 %. However, this model has almost the same False Positives than False Negatives, both bigger than zero, so if we use the 30-epoch model, whatever the given prediction we will not be sure that it is not an error. However, with the 40-epoch model, if an image is classified as normal, it's possible to be a false negative given by the model, but if an image is classified as abnormal, and knowing that the model does not give false positives, we will be sure that this patient has tuberculosis and no more tests will be needed. Then, if this model is used to diagnose, healthy patients will need further tests than this one, but 50 % of patients with Tuberculosis wont need further tests to confirm their diagnose.

Finally, as seen in table (4.4) the ResNet50 model trained for 50 epochs has only one false negative. If this model would keep these results when tested with more images, then it could be used in real diagnosis, because the model could be used as a fast test to ensure a patient does not have tuberculosis. However, this model has poor accuracy, so even assuming the existence of the only false negative, the performance of this model would be also poor.

4.2.2. Ensembles

Finally we are going to look for a better model in the 3 CNN's models ensembles. As explained in chapter (3) we will take our 3 best models (ResNet50, InceptionV3 and VGG19) to build 2 ensembles, which best results are shown in table (4.6) with their confusion matrix.

As we can see, the ensembles have better performance than CNNs stand-alone models, not only in overall accuracy, improving it to a maximum of 86 %, but also we find models without False Positives and less false negatives than the best CNN-alone model. Looking at table (4.6), we find that the best model is the 30-epoch ResNet50, the 10-epoch InceptionV3 and the 50-epochs VGG19 models, giving us no false positives and misclasifying only 12 out of 40 Abnormal Images, which means that we can avoid further tests in a 70 % of patients that really suffer from tuberculosis, so assuming that a 50 % of suspects of being tuberculosis sufferers really have the disease, and the other 50 % is healthy, using this model would reduce the total number of further tests in a 35 %.

Ensembler Type	**Epochs**		**Confusion Matrix**			**Total Accuracy**
Ensembler 1	**ResNet50**	50		**Normal**	**Abnormal**	0.8642
	InceptionV3	10	**Normal**	39	2	
	VGG19	50	**Abnormal**	9	31	
Ensembler 1 & 2	**ResNet50**	40		**Normal**	**Abnormal**	0.7901
	InceptionV3	40	**Normal**	41	0	
	VGG19	50	**Abnormal**	17	23	
Ensembler 1 & 2	**ResNet50**	30		**Normal**	**Abnormal**	0.8519
	InceptionV3	10	**Normal**	41	0	
	VGG19	50	**Abnormal**	12	28	
Ensembler 2	**ResNet50**	30		**Normal**	**Abnormal**	0.8272
	InceptionV3	40	**Normal**	41	0	
	VGG19	50	**Abnormal**	14	26	

Tabla 4.6: Ensemble best models confusion matrix and total accuracy by epochs of individual algorithms

5

Conclusions

This project has been focused on getting a better understanding of some Deep learning's algorithm such as the Convolutional Neural Networks and comparing their performance with a Machine Learning algorithm like the Support Vector Machine. For this reason I have carried out a theoretical survey of the algorithms and the processing techniques that are necessary to improve the performance of the algorithms, and the differences that there are between them.

The second goal was the study the applications that these algorithms could have in a field like medicine. For this I have designed a project which consists on the diagnosis of Tuberculosis by classifying chest X-Ray images, trying to find the model that fits better with the chosen dataset. From the few number of medical images that use to be available to the criterion that we would use to test a model and measure its performance in medical diagnosis I went through the different problems faced by this kind of algorithms in medicine, and how they are solved.

First we realize data augmentation on the chosen dataset by adding mirrored images, images with some random noise... and we have passed every image for some contrast enhancement algorithms such as the histogram equalization or the contrast limited adaptive histogram equalization (CLAHE) to prepare the dataset before start training the models. Then we have trained several CNN and SVM models, and some ensembles with the best CNN models. Finally we analyzed the model's results according to what would be necessary to use that models in real diagnosis, finding the models with the highest performance.

Future Work

There are several considerations to take into account to continue this work. Our models have been tested with only 81 images, so if one of the models were to be used we should find more images to test the models. If non further tests are performed then the statistical evidence given by the model would not be enough to use it in a real diagnosis service, because an error could be dangerous, leading to the patient's death. The models can be trained during more epochs, to check were the models' accuracy stops growing and they start overfitting the dataset.

Although we have found a good model without false positives, there would be more interesting to continue studying the ResNet50 model, because as we see in table (4.4), when trained during 50 epochs the number of false negatives is 1, which could lead to find a model without any false negatives. As explained in section (4), for diagnosing purposes is preferable a model without false negatives than without false positives.

Finally, to find a better model, as explained in section (2), we can remove the last layers of the CNN's models and enter the output in a classifier, such as a Support Vector Machine, to use them as a feature extractor for the classifier. We could use for this purpose the ResNet50 model trained during at least 50 epochs, because it was the model with less false negatives.

Bibliografía

[1] Rahul C. Deo. «Machine Learning in Medicine». En: *Circulation* 132.20 (2015), págs. 1920-1930. DOI: 10.1161/circulationaha.115.001593.

[2] Rafael C. Gonzalez y Richard E. Woods. *Digital Image Processing (3rd Edition)*. Pearson, 2007. ISBN: 9780131687288. URL: https://www.amazon.com/Digital-Image-Processing-Rafael-Gonzalez/dp/013168728X?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=013168728X.

[3] A. Walker R. Fisher S. Perkins y E. Wolfart. *Pixel Values*. Online; last acces 16-10-2018. 2003. URL: https://homepages.inf.ed.ac.uk/rbf/HIPR2/value.htm.

[4] Arthur COSTE. *Project 1 : Histograms*. Access Online; Last Access 17-10-2018. URL: http://www.sci.utah.edu/~acoste/uou/Image/project1/Arthur_COSTE_Project_1_report.html#Segmentation.

[5] D.W.Murray. *Computer Vision, Graphics, and Image Processing*. Vol. 39. Springer-Verlag, 1987, págs. 355-368. ISBN: 0734-189X.

[6] Connor Shorten. *Data Augmentation on Images*. Online; last access 17-10-2018. Abr. de 2018. URL: https://towardsdatascience.com/data-augmentation-and-images-7aca9bd0dbe8.

[7] F. Pedregosa y col. «Scikit-learn: Machine Learning in Python ». En: *Journal of Machine Learning Research* 12 (2011), págs. 2825-2830.

[8] Tristan Fletcher. *Support Vector Machines Explained*. Online; last access 20-10-2018. Dic. de 2008. URL: https://cling.csd.uwo.ca/cs860/papers/SVM_Explained.pdf.

[9] Juergen Schmidhuber. «Deep learning in neural networks: An overview». En: *Neural Networks* 61 (Jan 2015), págs. 85-117. DOI: 10.1016/j.neunet.2014.09.003.

[10] University of Copenhagen (DIKU) Steffen Nisse Department of Computer Science. *IMPLEMENTATION OF A FAST ARTIFICIAL NEURAL NETWORK LIBRARY (FANN)*. Online; last access 04-01-2018. 2003. URL: http://leenissen.dk/fann/report/node4.html.

[11] P Sibi, S Allwyn Jones y P Siddarth. «Analysis of different activation functions using back propagation neural networks». En: *Journal of Theoretical and Applied Information Technology* 47.3 (2013), págs. 1264-1268.

[12] Daniel Svozil, Vladimír Kvasnicka y Jir

í Pospichal. «Introduction to multi-layer feed-forward neural networks». En: *Chemometrics and Intelligent Laboratory Systems* 39.1 (1997), págs. 43-62. DOI: 10.1016/s0169-7439(97)00061-0.

[13] Raul Rojas. *Neural Networks A Systematic Introduction*. https://page.mi.fu-berlin.de/rojas/neural/. Springer-Verlag, 1996.

[14] Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[15] Sinno Jialin Pan y Qiang Yang. «A Survey on Transfer Learning». En: *IEEE Transactions on Knowledge and Data Engineering* 22.10 (2010), págs. 1345-1359. DOI: 10.1109/tkde.2009.191.

[16] Jason Yosinski y col. «How transferable are features in deep neural networks?» En: *Advances in Neural Information Processing Systems 27*. Ed. por Z. Ghahramani y col. Curran Associates, Inc., 2014, págs. 3320-3328. URL: http://papers.nips.cc/paper/5347-how-transferable-are-features-in-deep-neural-networks.pdf.

[17] Stefan Jaeger y col. «Two public chest X-ray datasets for computer-aided screening of pulmonary diseases». En: *Quantitative Imaging in Medicine and Surgery* 4.6 (2014). URL: http://qims.amegroups.com/article/view/5132.

[18] François Chollet y col. *Keras*. https://keras.io. 2015.

[19] Olga Russakovsky y col. «ImageNet Large Scale Visual Recognition Challenge». En: *International Journal of Computer Vision (IJCV)* 115.3 (2015), págs. 211-252. DOI: 10.1007/s11263-015-0816-y.

Apéndices

A

Code

For this project I have developed some python functions and python scripts to manage the data, process it, and train and test the algorithms. These are the libraries we will use for the project

Código A.1: Libraries Used

```
import os
import shutil
import cv2
import numpy as np
import pickle
import random
import sys

# CNN libraries
import keras
from keras.preprocessing import image
from keras.applications import resnet50, inception_v3, vgg16, vgg19
from keras.models import Model
from keras.layers import Dense, GlobalAveragePooling2D, Input
from keras.optimizers import Adam, SGD
import preProcess as pd
from keras.utils import np_utils

# SVM Libraries
from sklearn import svm
import sklearn
from sklearn.metrics import confusion_matrix
```

A.1. Managing the dataSet

First we separate our data between Normal Images, which will be renamed as **+i.png** and Abnormal Images, renamed as **_j.png** where $i \in (0, \#(NormalImages))$ and $j \in (0, \#(AbnormalImages))$. We also resized the images to 224x224. To do this, we used the pyhton script (A.2) with the original images, located at **Source/Normal** and **Source/Abnormal**, and set the destination folders to **Normal/** and **Abnormal/**.

Código A.2: Managing the Original DataSet

```
if not os.path.exists('Normal'):
    os.mkdir('Normal')

if not os.path.exists('Abnormal'):
    os.mkdir('Abnormal')

filelist_normal=os.listdir('Source/Normal/')
filelist_abnormal=os.listdir('Source/Abnormal/')

for i in range(len(filelist_normal)):
    img = cv2.imread('Source/Normal/'+filelist_normal[i])
    resized_image = cv2.resize(img, dsize=(224,224), interpolation=
        ↪ cv2.INTER_CUBIC)
    cv2.imwrite('Normal/_'+str(i)+'.png', resized_image)

for i in range(len(filelist_abnormal)):
    img = cv2.imread('Source/Abnormal/'+filelist_abnormal[i])
    resized_image = cv2.resize(img, dsize=(224,224), interpolation=
        ↪ cv2.INTER_CUBIC)
    cv2.imwrite('Abnormal/_'+str(i)+'.png', resized_image)
```

A.2. Processing Images

Before passing the images to the Machine Learning Algorithms, we will process them to perform some data augmentation and prepare them to get a better performance with the algorithms.

First we will separate the images between Training and Testing Images. Then for each training image we will add another one by mirroring the original,

Código A.3: Basic Processing Functions

```
def mirror_image(img):
  horizontal_img = cv2.flip( img, 1 )
  return horizontal_img

def histogram_equalization(img):
  equ = cv2.equalizeHist(img)
  return equ

def CLAHE(img, clipLimit, tileGridSize):
  clahe = cv2.createCLAHE(clipLimit, (tileGridSize,tileGridSize))
  cl1 = clahe.apply(img)
  return cl1

def add_gaussian_noise(image_in, noise_sigma=5):

        temp_image = np.float64(np.copy(image_in))
        h = temp_image.shape[0]
        w = temp_image.shape[1]
        noise = np.random.randn(h, w) * noise_sigma
        noisy_image = np.zeros(temp_image.shape, np.uint8)
        if len(temp_image.shape) == 2:
                        noisy_image = temp_image + noise
        else:
                        noisy_image[:,:,0] = temp_image[:,:,0] +
                           ↪ noise
                        noisy_image[:,:,1] = temp_image[:,:,1] +
                           ↪ noise
                        noisy_image[:,:,2] = temp_image[:,:,2] +
                           ↪ noise
        return noisy_image
```

Código A.4: Processing All Images (i)

```
def process_Images(trainPercentage=0.9, clahe_option = False,
    ↪ clipLimit=2, tileGridSize=8):

        originalNormalImg = []
  originalAbnormalImg = []

        # Open Normal and Abnormal Images
        for i in range(len(os.listdir(os.path.join('Normal')))):
                img_name = os.path.join('Normal',"+"+str(i))+".png"
    originalNormalImg.append(cv2.imread(img_name, 0))
  for i in range(len(os.listdir(os.path.join('Abnormal')))):
    img_name = os.path.join('Abnormal',"_"+str(i))+".png"
    originalAbnormalImg.append(cv2.imread(img_name, 0))

        # Split Train and Test Data
  trainNormalImg = originalNormalImg[:int((len(originalNormalImg)*
     ↪ trainPercentage))]
  testNormalImg = originalNormalImg[int((len(originalNormalImg)*
     ↪ trainPercentage)):]
  trainAbnormalImg = originalAbnormalImg[:int((len(
     ↪ originalAbnormalImg)*trainPercentage))]
  testAbnormalImg = originalAbnormalImg[int((len(
     ↪ originalAbnormalImg)*trainPercentage)):]

        # Mirroring Train Images
        for i in range(len(trainNormalImg)):
    trainNormalImg.append(mirror_image(trainNormalImg[i]))
  for i in range(len(trainAbnormalImg)):
    trainAbnormalImg.append(mirror_image(trainAbnormalImg[i]))

        # Apply CLAHE or Histogram Equalization
        if clahe_option == True:
                for i in range(len(trainNormalImg)):
                        trainNormalImg[i] = CLAHE(trainNormalImg[i
                           ↪ ], clipLimit, tileGridSize)
                for i in range(len(trainAbnormalImg)):
                        trainAbnormalImg[i] = CLAHE(
                           ↪ trainAbnormalImg[i], clipLimit,
                           ↪ tileGridSize)
                for i in range(len(testNormalImg)):
                        testNormalImg[i] = CLAHE(testNormalImg[i],
                           ↪ clipLimit, tileGridSize)
                for i in range(len(testAbnormalImg)):
                        testAbnormalImg[i] = CLAHE(testAbnormalImg[
                           ↪ i], clipLimit, tileGridSize)
```

Código A.5: Processing All Images (ii)

```
else:
                for i in range(len(trainNormalImg)):
                        trainNormalImg[i] = histogram_equalization(
                            ↪ trainNormalImg[i])
                for i in range(len(trainAbnormalImg)):
                        trainAbnormalImg[i] =
                            ↪ histogram_equalization(
                            ↪ trainAbnormalImg[i])
                for i in range(len(testNormalImg)):
                        testNormalImg[i] = histogram_equalization(
                            ↪ testNormalImg[i])
                for i in range(len(testAbnormalImg)):
                        testAbnormalImg[i] = histogram_equalization
                            ↪ (testAbnormalImg[i])

        # For each image in the training set, add another image
            ↪ with random Gaussian Noise with a probability of 25\%
for i in range(len(trainNormalImg)):
                if np.random.rand() > 0.75:
                        trainNormalImg.append(add_gaussian_noise(
                            ↪ trainNormalImg[i], 5))
for i in range(len(trainAbnormalImg)):
  if np.random.rand() > 0.75:
    trainAbnormalImg.append(add_gaussian_noise(trainAbnormalImg[i
       ↪ ], 5))

        # Create Labels as a numpy array: 0 for Normal images, 1
            ↪ for Abnormal images
        trainNormalLabel = np.zeros(len(trainNormalImg))
trainAbnormalLabel = np.ones(len(trainAbnormalImg))
testNormalLabel = np.zeros(len(testNormalImg))
testAbnormalLabel = np.ones(len(testAbnormalImg))

        # Join each image with it's label
trainNormalLabeled = list(zip(trainNormalImg, trainNormalLabel.
    ↪ copy()))
trainAbnormalLabeled = list(zip(trainAbnormalImg,
    ↪ trainAbnormalLabel.copy()))
testNormalLabeled = list(zip(testNormalImg, testNormalLabel.copy
    ↪ ()))
testAbnormalLabeled = list(zip(testAbnormalImg, testAbnormalLabel
    ↪ .copy()))
```

Código A.6: Processing All Images (iii)

```
        # Merge Normal and Abnormal Data
        trainLabeled = trainNormalLabeled + trainAbnormalLabeled
testLabeled = testNormalLabeled + testAbnormalLabeled

        # Shuffle Data
random.shuffle(trainLabeled)
random.shuffle(testLabeled)

        # Split Image and Labels
        aux = [list(t) for t in zip(*trainHistLabeled)]
trainImg = aux[0].copy()
trainLabel = aux[1].copy()
aux = [list(t) for t in zip(*testHistLabeled)]
testImg = aux[0].copy()
testLabel = aux[1].copy()

        # Transform each image in a 3-channel image (Only for CNN's
           ↪ models, not for SVM)
for i in range(len(trainImg)):
  trainImg[i] = np.stack((trainImg[i],)*3, axis=-1)
for i in range(len(testImg)):
               testImg[i] = np.stack((testImg[i],)*3, axis=-1)

return np.array(trainImg), trainLabel, np.array(testImg),
   ↪ testLabel
```

A.3. Convolutional Neural Netowrks

The first step is add process the images and create the keras data structure and fit our data into it.

Código A.7: Preparing the Data

```
# Process the data
x_train, x_label, x_test, x_test_label = process_Images(0.9)

# Set the CNN pparameters
batch_size = 32
epoch = 10

# Create the Keras data structure
datagen = keras.preprocessing.image.ImageDataGenerator(
    ↪ samplewise_center=True,
        samplewise_std_normalization=True,
        rotation_range=0,
        width_shift_range=0.0,
        height_shift_range=0.0,
        brightness_range=None,
        shear_range=0.0,
        zoom_range=99.0,
        channel_shift_range=0.0,
        fill_mode='nearest',
        cval=0.0,
        horizontal_flip=False,
        vertical_flip=False,
        rescale=1/255,
        preprocessing_function=None,
        data_format="channels_last",
        validation_split=0.1)

# Fit our data and labels into it
datagen.fit(x_train)
datagen.fit(x_test)
train_Label = np_utils.to_categorical(x_label, 2)
test_Label = np_utils.to_categorical(x_test_label, 2)
```

Finally we will build our CNN models by loading the models ResNet50, InceptionV3 or VGG19 , loading the 'imagenet' weights, removing the last layer (prediction layer) and adding other prediction layer which classify in 2 classes.

Código A.8: Training the Model

```
# Choose one of the base models
#base_model = resnet50.ResNet50
#base_model = inception_v3.InceptionV3
#base_model = vgg19.VGG19

# Load the imagenet weights
base_model = base_model(weights='imagenet', include_top=False)

# Replace the default classifier with another that classify between
    ↪ 2 classes
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(num_classes, activation='sigmoid')(x)
model = Model(inputs=base_model.input, outputs=predictions)
for layer in base_model.layers:
                layer.trainable = True

# Compile the model with the chosen parameters
model.compile(loss='binary_crossentropy',
                                        optimizer=Adam(lr=0.0001),
                                        metrics=['acc'])

# Train the model
model.fit(x_hist_train, train_Hist_Label,
                        epochs=epoch,
                        validation_split=0.1,
                        batch_size=batch_size,
                        shuffle=True,
                        verbose = 1)

# Save the trained model to disk
fileName = "./resNet50_Fine_Tunes_Hist_"+str(epoch)+".h5"
model.save(fileName)

# Get the prediction's array, which is a list of pairs, being the
    ↪ first element
# the predicted probability of belonging to class 0 and the second
    ↪ the
# probability of belonging to class 1
score = model.predict(x_hist_test))
```

A.4. Support Vector Machine

We will build or SVM models using the library **scikit-learn**. We will process the data the same way than for the CNN's models, excepting that the SVM algorithms get 1-channel images, so we will flatten the each image from 224x224 to 1x50176.

Código A.9: SVM Model

```
def create_model(trainingSet, trainingLabel, kernel, degree=2,
    ↪ probability=True):

        clf = svm.SVC(gamma="auto", probability=probability, kernel
            ↪ =kernel, degree=degree)
        clf.fit(trainingSet, trainingLabel)
        return clf

def test_model(clf, testSet, testLabel):

    result = clf.predict(testSet)
    conf = confusion_matrix(y_true=testLabel, y_pred=result)
    return conf, result

x_train, x_label, x_test, x_test_label = process_Images(0.9)

train = []
test = []

# Flatten each image from 224x224 to 1x50176
for i in range(len(x_hist_train)):
    train.append(x_hist_train[i].flatten())
for i in range(len(x_hist_test)):
    test.append(x_hist_test[i].flatten())

model = create_model(train, x_label, 'poly', degree=7)
conf, result = test_model(model, test, x_test_label)
```

A.5. Ensembles

As explained in chapter (3), we will build 2 ensembles:

Código A.10: Ensembles

```
def ensemble_1(prediction_model_1, prediction_model_2,
    ↪ prediction_model_3):

        result = []
        for i in range(len(prediction_resnet)):
                score = prediction_model_1[i]+prediction_model_2[i
                    ↪ ]+prediction_model_3[i]
                if score[0] > score[1]:
                        result.append(0)
                else:
                        result.append(1)
        return result

def get_result_ensembler_2(prediction_model_1, prediction_model_2,
    ↪ prediction_model_3):

    results = []
    for i in range(len(prediction_resnet)):
        score = 0
        if prediction_model_1[i][0] <= prediction_model_1[i][1]:
            score = score + 1
        if prediction_model_2[i][0] <= prediction_model_2[i][1]:
            score = score + 1
        if prediction_model_3[i][0] <= prediction_model_3[i][1]:
            score = score + 1
        if score >= 2:
            results.append(1)
        else:
            results.append(0)
    return results
```